Faculty of Natural and Applied Sciences Journal of Computing and Applications
Print ISSN: 3026-8133
www.fnasjournals.com
Volume 2; Issue 2; March 2025; Page No. 1-9.

A Framework for Early Detection of Agile Software Development Project Failures

***[1]Nanwin, D.N., [2]Agaji, I., [2]Ogala, E., & [2]Gbaden, T.**
[1]Department of Computer Science, Ignatius Ajuru University of Education, Port Harcourt, Nigeria
[2]Department of Computer Science, Joseph Sarwuan Tarka University Makurdi, Benue State, Nigeria

***Corresponding author email**: kakusman@yahoo.com

Abstract
In the realm of software development, the early identification of project failures is crucial for ensuring project success and minimizing risks. This study developed an early detection framework using various machine learning models to anticipate potential failures. Agile software projects were used for the study. The framework employed a range of machine learning models including decision tree, bagging classifier, weighted bagging classifier, random forest classifier, weighted random forest classifier, decision tree estimator, and bagging estimator. These models are trained and tested using a dataset comprising 13,238 observations from 12 different software companies, each with 15 variables relevant to project performance and outcomes. Initial training of the different models yielded promising results, with performance ranging between 45% to 55% accuracy during testing. Despite attempts to enhance the model's performance, including refinement of features and algorithms, there were no significant improvements observed. The evaluation results highlight the need for further refinement and optimization of the models used in the framework. In conclusion, while the decision tree classifier, bagging classifier, and random forest exhibited outstanding performance in the training results, the overall evaluation suggests that more work is required to improve the effectiveness of the early detection framework for Agile software project failures. Further research and refinement of the models are necessary to enhance accuracy and reliability in identifying potential project failures early in the Agile software development lifecycle.

Keywords: Agile, Framework, Project Failure, Machine Learning, Software Development.

Introduction

In the dynamic landscape of software development, Agile methodologies have emerged as a popular approach due to their flexibility and adaptability. Despite its advantages, Agile projects are not immune to failure. According to Osegi et al. (2018) recognizing the importance of early intervention, there is need to presents an efficient framework tailored for the early detection of failures for Agile software development projects which will empower teams with proactive measures to mitigate risks, optimize performance, and ultimately enhance project success rates. A structure placed on the creation of a software product is known as a software development process. A framework is a physical or conceptual structure that is meant to support and direct the construction of something that enlarges it into a more useful form (Sommerville, 2011). The problem of software development project failure detection is not a new area of research. However, its early detection using machine learning has sprung up a number of renewed interests by system engineers and scientists. One primary reason is the special property of machine learning which makes the machines (computers in this case) more useful in the process of software development. However, due to numerous ML techniques, the matter of choice of techniques is still a primary issue. Some recent researches include rigorous evaluation of several benchmark ML techniques with the hope that a single universal technique can be identified. Thus, focus is made on evaluating single MLTs or ensembles on some datasets or a particular dataset. Indeed, it has been found that decision tree ensembles (Random Forests or simply RFs) are best candidates (Zhang & Suganthan, 2014). The problem associated with other software project failure detection framework is that they are reactive; that is software project fault is already introduced into the system before the faults are detected. They did not consider a central software fault detection framework for the system, also not all the parameters that could lead to software project failure are considered by them. They do not give better and accurate results. They cannot easily identify trend and

Cite this article as:
Nanwin, D.N., Agaji, I.,Ogala, E., & Gbaden, T.(2025). A framework for early detection of agile software development project failures. *FNAS Journal of Computing and Applications, 2*(2), 1-9.

patterns and cannot handle multi-dimensional and multi-variety data. The work develops an efficient framework for early detection of agile software development project failures through the identifying the fundamental causes of software development project failures, the development of a novel framework for early detection of software development failures, the application of an ensemble learning model: machine learning technique for analysis, classification and detection of the software failure signs and a comparative analysis using relevant evaluation metrics. Dauda et al. (2021) highlight that software failure arises when a developed project diverges from software project failures on organizations, including financial losses and damage to reputation. Through a comprehensive analysis of related literature, the study sheds light on factors contributing to project failure, including schedule pressures, inadequate requirements, skill deficiencies, unrealistic expectations, and task allocation issues. The authors stress the importance for both new and existing organizations to comprehend these causes and implement realistic measures to ensure the effectiveness of their software projects. Egbokhare (2014) investigates the causes of software and information technology project failures within software development organizations in Nigeria. The work employed a descriptive research method which examined 20 randomly selected organizations involved in software development across the country. The study reveals that many of these organizations do not adhere to structured software development methodologies, which contributes to project failures. Additionally, the research identifies various other factors that contribute to software development failures in Nigeria. Chillar and Sharma (2019) introduced a T-model encompassing 45 quality metrics based on empirical reviews of the root causes of software failures. The authors emphasized the importance of understanding recent software development methods to comprehend contemporary failures. They discussed the technological evolution over time and tested the progression of non-functional parameters, particularly focusing on security and performance, which constitute non-functional software requirements. Eberendu (2015) conducted an evaluation of software project abandonment and failure in tertiary institutions in Nigeria using an investigative and qualitative method. They collected data through questionnaires from heads of computer units in government-owned tertiary institutions in the South-East and South-South regions of Nigeria. Their findings revealed that the reasons for software failure and abandonment are complex and multifaceted, defying simple explanations. Nevendra and Singh (2021) employed an enhanced convolutional neural network, a deep machine learning technique, for predicting software defects. The work detected defects using historical datasets and was applied to nineteen open-source software defect datasets. The evaluation involved various metrics, demonstrating significant performance. Furthermore, the Scott-Knot ESD (Electrostatic Discharge) test was conducted to validate the effectiveness of the approach, showcasing its potential for accurate defect prediction in software development projects.

Janssen (2019) reviewed literature on predicting software project success rates before initiation using Bayesian Classifier and logistic regression models. The study emphasized defining research goals and gathering assumptions from experts. It identified critical success factors and metrics, validating them through expert interviews and surveys. The study provided a framework for developing a prediction model pending suitable dataset availability. Alonso et al. (2011) developed several machine learning models, including decision tree, Linear and Quadratic discriminant analysis, Naïve Bayes, Support Vector Machine, K-nearest Neighbor, and random forest algorithms, to predict system states using monitoring system metrics. This proactive approach enables automatic software transformation triggered by predicted anomalies. The study found that the Random Forest algorithm had a validation error of less than 1%, outperforming other machine learning algorithms evaluated. Additionally, the work integrated Lasso Regularization technique with machine learning classifiers to automatically reduce the number of monitored parameters needed for anomaly prediction, achieving up to a 60% reduction in the best case. The framework, validated in an ecommerce environment with Apache Tomcat and MySQL database servers, demonstrates the efficacy of machine learning and Lasso regularization techniques in proactive system state prediction and anomaly detection. Peng et al. (2015) conducted an empirical study on software project failure prediction using a simplified metric set. The work built software defect prediction models employing six well-known classification algorithms: J48, Logistic Regression (LR), Naïve Bayes (NB), Decision Table (DT), Support Vector Machine (SVM), and Bayesian Network (BN) implemented in Weka. Three types of predictors were constructed based on the size of the software metric set in three scenarios, validated using Top-k metrics and statistical methods.

They aimed to minimize the Top-k metric subset by removing redundant metrics and tested the stability of the minimum metric subset with one-way ANOVA tests. The study encompassed 34 releases of 10 open-source software projects available at the PROMISE repository. Results indicate that both the top-k metrics and minimum metric subsets provide acceptable performance compared to benchmark predictors, suggesting that simplified metric sets are effective, especially in resource-constrained situations. Ebubeogu and Lee (2017) developed simple linear regression and multiple linear regression models to predict software project failures. These models utilized predictor variables such as defect density, defect velocity, and defect introduction time, derived from defect acceleration, to forecast the

Cite this article as:
Nanwin, D.N., Agaji, I.,Ogala, E., & Gbaden, T.(2025). A framework for early detection of agile software development project failures. *FNAS Journal of Computing and Applications, 2*(2), 1-9.

total number of project failures in software development. The work also devised a framework consisting of two phases: data preprocessing and data analysis, to predict software project failure. The study found that defect velocity exhibited the strongest correlation coefficient of 0.98% in predicting the number of defects. However, while the work aimed to identify data preprocessing and analysis phases as potential points for defect or failure detection, it overlooked the fact that software failure can occur at any phase in the software development life cycle.

Methodology

The research study adopts a dual holistic approach in its methodology. Initially, it employs an agile approach, specifically Crystal Clear methodology, emphasizing frequent delivery, reflective improvement, and osmotic communication, with personal safety as core guiding principles. The Crystal Clear Agile Methodology (CCAM) ensures that strategies, techniques, and team roles at each stage of the project iteration are carefully considered. Furthermore, the systems methodology is integrated to promote developmental efficiency and enhance the habitability of software engineering project conventions. Overall, this comprehensive approach addresses the requirements of the Agile manifesto and facilitates the development of an effective early software project failure detection framework. The architecture of the framework is as shown in Figure 1

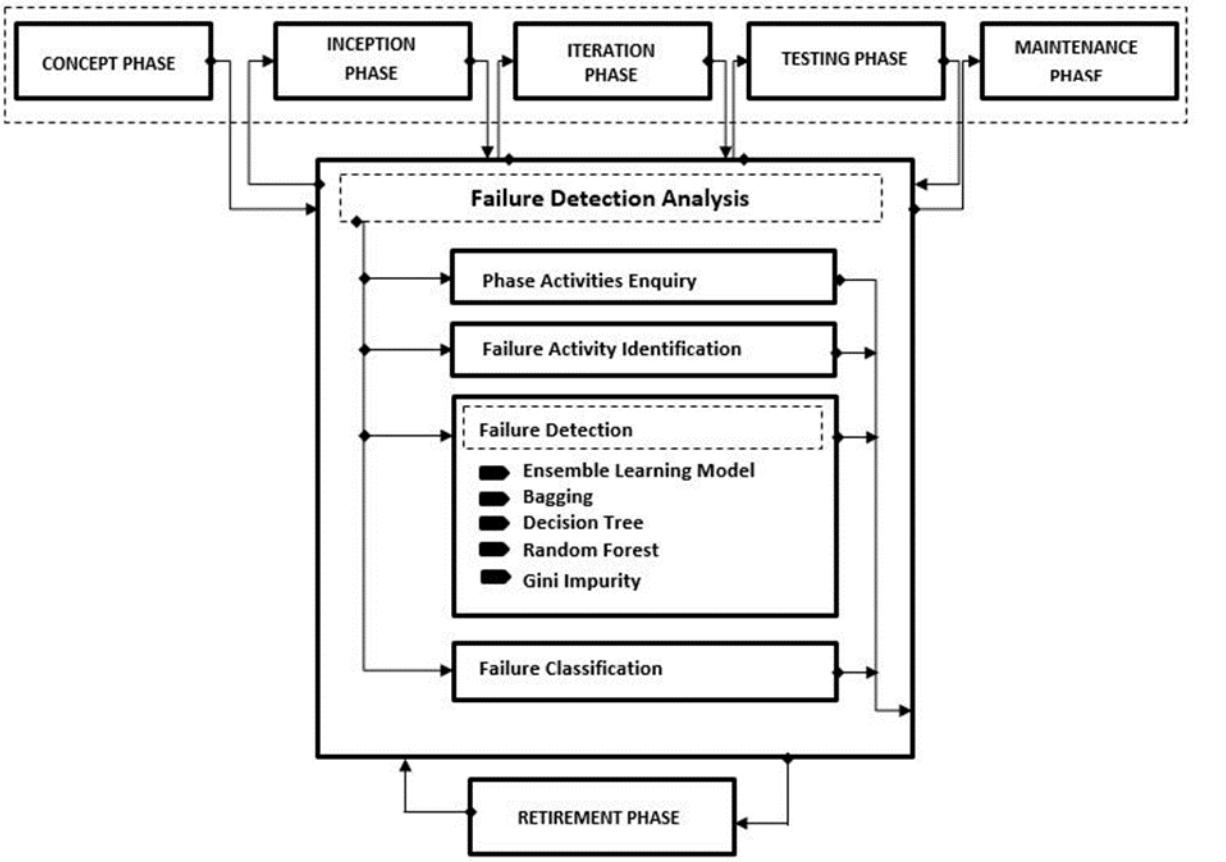

Figure 1: Architecture of the Proposed Framework

Figure 1 gives a diagrammatical presentation of the developed general framework with the intent that software development project failure can come as early as from concept/ scope-out analysis phase and to the point of maintenance/ support phase. It is important to note that most software development project failed because of inadequate scope-out analysis. This illustration has been able to include failure detection analysis check at the end of every phase before entering into the next phase of the Agile software development life cycle. This means that, for every phase, a failure analysis is done to identify early software development project failures and to detect and classify these failures as possible reason the software development project will fail. The output of each phase is subjected to failure detection analysis before starting the next phase. This detection strategy follows through with each phase of

Cite this article as:
Nanwin, D.N., Agaji, I.,Ogala, E., & Gbaden, T.(2025). A framework for early detection of agile software development project failures. *FNAS Journal of Computing and Applications, 2*(2), 1-9.

the project development life cycle explaining what transpires or how the software development project failure is detected during the failure detection analysis. Altogether, there are four (4) stages in which Agile phases are scrutinized for potential indicators of software development project failure.

The Phase Activity Enquiry checkmates the various activities carried out in each phase of the SDLC. During this enquiry process, eight (8) early signs for software development project failure were used as a benchmark to analyze the phases. These signs include working in parallel, big team from the beginning, documentation piled-up, No Ask – No Tell policy, requirements clarifications delayed, crucial tasks outsourced, integration of multiple products/ technologies and management ignorance. These signs or signals are checked throughout the phases of the development life cycle. Documents are compiled to help for a clearer analysis of what is being transpired in the development process of the system.

This stage is very important in the sense that, data are being compiled in this stage. The data compiled are documented for analytical purposes from which indications for software development project failures can be identified. The data gathering method used here is through questionnaire and interviewing (interacting with) key players in the industry. The documentation of the system requirement, functions, features (what must be done) and tools to be used for the development of the software project is reviewed and documented. Failure Activity Identification is done based on the documented enquiries gotten from the phase activities enquiry section. The task here is to be able to identify these failure signs as each phase of the software development is going on and to scale it with respect to certain percentage. From these enquiries and documentation, the activities of the software development project are identified and if there is any failure signal, they are detected. From the failure activity enquiry, some information were recorded during the enquiry or enquiries. The combination of machine learning algorithms, were used to perform a software failure analysis and based on the result(s) of the analysis, a decision was reached. When these failure signs are identified, corrections can be done immediately. But, that will be determined by the percentage of occurrence of that sign or signal during the software development process. Every step of the software development project has a target and tools required to accomplish that phase. Failure of the software development project is detected in Failure Detection early in the development process. It is the main idea of this framework; to be able to detect software development project failure before it is too late and one cannot remedy the situation. This encourages software development project continuity and general success of a software development project.

Failure detection is possible after the various failure signals in each phase have been identified and analyzed. Looking at their statistics, one can spot the red flag and at what aspect of the Agile software development lifecycle phases there is likely to be software development failure and correction measure or due attention can be given to that phase. The application of Ensemble Learning, a combination of machine learning models were used to analyze the outcome of the identified signals during the phase activity enquiry and failure activity identification. These machine learning techniques were used to perform the failure detection analysis through the process. The data were fed into the algorithms of each machine learning technique and failure analysis done on the output result of each of the algorithms. With the results gotten, conclusions were reached on whether the software development project is going to be successful or its going to fail. The three models follows the same pattern in evaluating the dataset collected for this purpose. Bagging, an ensemble learning method (referred to as bootstrap aggregation) was used to reduce variances in a noisy dataset. It does that by randomly sampling the training data which is selected with replacement. In doing this, data points can be chosen more than ones during model building and evaluation. The decision tree, an ensemble learning model uses categorization method to make predictions. The supervised learning model is trained and tested on a set of data containing the categorization needed and random forest model uses multiple decision trees that are merged together to make accurate prediction. This is with the idea that, multiple uncorrelated models (i.e. individual decision tree) can perform much better when grouped together than when they are used alone.

Detailed System Design

The study will adopt the Universal Modeling Language (UML) in the description of the system. The Sequence diagram is used to demonstrate the functionality of the framework. The proposed framework for early detection of software development project failures has been represented in the sequence diagram as shown in Figure 2.

Cite this article as:
Nanwin, D.N., Agaji, I.,Ogala, E., & Gbaden, T.(2025). A framework for early detection of agile software development project failures. *FNAS Journal of Computing and Applications, 2*(2), 1-9.

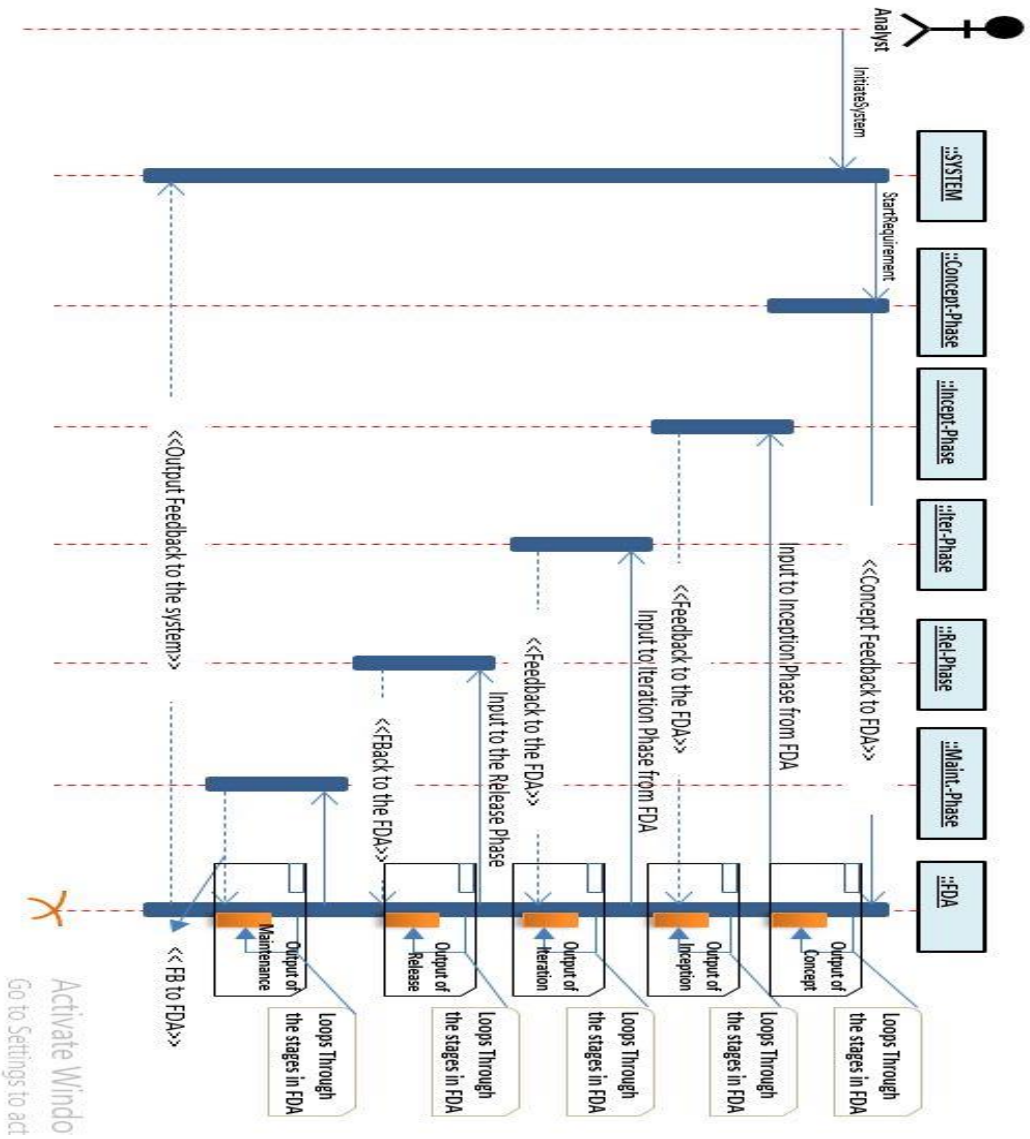

Figure 3: Sequence Diagram of the Proposed System

In Figure 2 after the system has been initiated, the output (which are the scope, requirements, and the supported features) of the concept/ scope-out phase analysis is forwarded to the Failure Detection Analysis (FDA) to check for the possible failure signs and the feedback is sent in as input to the next phase (Inception/ initial sprint Phase). The output (which include the selection of the team, user interface design, and architecture) of the inception/initial sprint phase is sent back to the Failure Detection Analysis for analysis going through the Phase Activity Enquiry (PAE) which checkmates the various activities that are carried out in each phase of the Agile software development life cycle (ASDLC), Failure Activity Identification (FAI) identify the various early failure signs at each phase of the ASDLC, Failure Detection (FD) help to detect the software project failure early in the development process and Failure Classification (FC) help to classify the early failure signs according to their various headings. This process is followed for each of the phases of the Agile SDLC as illustrated in Figure 2. The output of each phase always loops through the failure detection analysis steps before it is fed back to the next phase.

The output of phase 1 – Concept includes the software development project requirements, feasibility study report of the system and features to be added in the system. Possible failure signs could be the ambiguity of the requirements and features of the system. Phase 2 – Inception, its output may include built project team, mock-up user interface and

Cite this article as:
Nanwin, D.N., Agaji, I.,Ogala, E., & Gbaden, T.(2025). A framework for early detection of agile software development project failures. *FNAS Journal of Computing and Applications, 2*(2), 1-9.

tools to use for development. Some possible failure signs could be wrong team selection, ambiguous user interface and choosing the wrong tools for development. For phase 3 – Iteration, the first iteration/sprint working software product, codification of the requirements and product functionalities. A failure signal here could be when the design is not meeting up with product functionalities and requirements. The phase 4 – Release has to do with quality assurance test result, bugs and defect detection and fixing, trained users and proper documentation presentation. Some failure signs could be when the product is not meeting up with standards (i.e. the quality of product), bugs and defects that cannot be fixed, not properly trained users and documentation. Phase 5 – maintenance, the output of this phase are the deployed system, ongoing support to keep system running smoothly and upgrades. The failure sign here is the lack of support and continuous training of users. The output of retirement phase which is phase 6 send notification to user and an end - of – life activities and removal of support. Whereas this may not really mean much, it is good to retire a project well to avoid issues that may result from improper information dissemination to users. Lack of this may result in legal battle between users and the software product developers.

Results

The results displayed in Table 1 and Table 2 were obtained from the machine learning evaluation metrics outcome of the dataset from 12 different organizations. The dataset which contains information of software project failures of these organizations has over 13,000 instances with 15 variables which include 14 independent variable and 1 dependent variable. To undertake detection analysis and assess the model's efficacy for software project failure, the data was split into training and testing sets, each containing all the information necessary for detection analysis. In this investigation, the decision tree model was utilized. The performance of the experiment is validated using the performance metric accuracy, recall, precision and F1-Score. The machine learning models were trained on dataset and the performance evaluation result of the various model are shown in Table 1 and Table 2 with the corresponding graphical representation of the model shown in Figure 3 and Figure 4.

Table 1: Evaluation and Performance of the Different Models

	Decision Tree	Bagging Classifier	Weighted Bagging Classifier	Random Forest Classifier	Weighted Random Forest Classifier	Decision Tree Estimator	Bagging Estimator
Accuracy	0.50806	0.507553	0.504783	0.490181	0.484391	0.497231	0.497231
Recall	0.47798	0.421266	0.33519	0.481013	0.430886	1.0	1.0
Precision	0.50562	0.505775	0.50304	0.487179	0.479437	0.497231	0.497231
F1	0.49141	0.459669	0.402309	0.484076	0.453867	0.6642	0.6642

Cite this article as:
Nanwin, D.N., Agaji, I.,Ogala, E., & Gbaden, T.(2025). A framework for early detection of agile software development project failures. *FNAS Journal of Computing and Applications, 2*(2), 1-9.

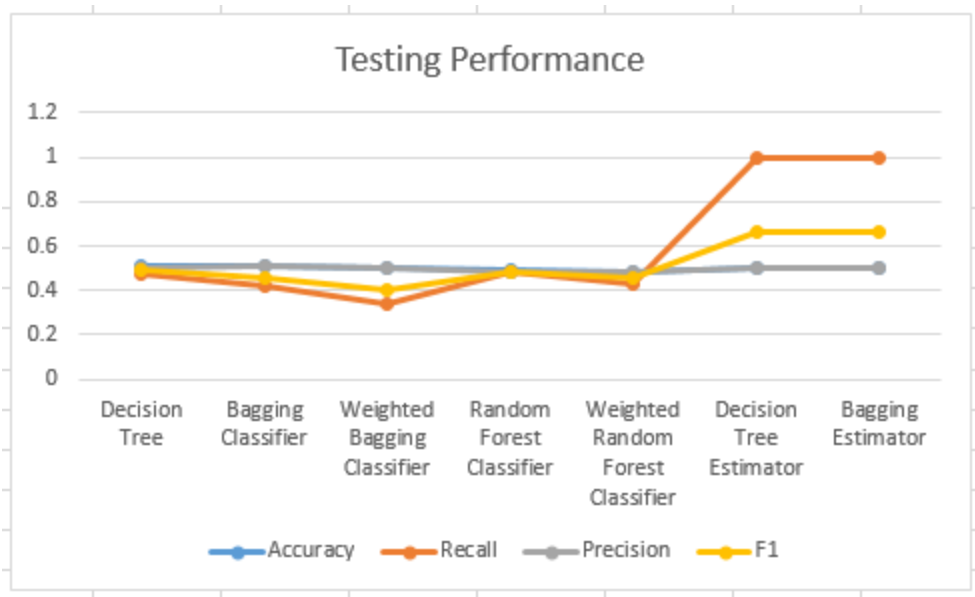

Figure 3: Graphical Representation of the Evaluation Result

Based on the performance results of the different models shown in Table 1, the Decision Tree Estimator and Bagging Estimator have perfect recall (100%), which means they are excellent at identifying all instances of project failure. This is crucial for early detection, as missing any potential failures could be detrimental. However, their accuracy is lower compared to other models, which indicates they might be less reliable in terms of overall correctness. Despite this, their ability to catch all potential failures makes them highly valuable in the context of early detection. The decision tree, bagging classifier and weighted random forest classifier had a moderate performance. These models have moderate performance with respect to recall, accuracy, and F1 score. They are better than some of the weaker models but do not excel in any particular metric. While they are better at precision and accuracy than the top-performing models, their lower recall means they might miss some project failures. This could be a disadvantage for early detection where catching as many failures as possible is crucial. The weighted bagging classifier and random forest classifier models show the lowest recall and overall performance. The Weighted Bagging Classifier, in particular, has very low recall, meaning it fails to detect a significant portion of actual failures. The Random Forest Classifier has better recall than the Weighted Bagging Classifier but still falls short compared to the top-performing models. For an efficient framework aimed at early detection of software development project failures, focusing on models with high recall (such as the Decision Tree Estimator and Bagging Estimator) is crucial to ensure that all potential issues are identified. The model's hyper parameters were fine-tuned to improve the outcome of the various models. The result of the fine-tuned models are shown in Table 2 and the corresponding graphical representation is shown in Figure 3.

Table 2: Evaluation Performance After Fine-Tuning

	Decision Tree	Bagging Classifier	Weighted Bagging Classifier	Random Forest Classifier	Weighted Random Forest Classifier	Decision Tree Estimator	Bagging Estimator
Accuracy	0.72190	0.79433	0.74698	0.81364	0.852	0.79509	0.79434
Recall	0.76113	0.78126	0.83270	0.83777	0.85753	0.64375	0.78126
Precision	0.72112	0.60854	0.81020	0.80215	0.82801	0.65909	0.70854
F1	0.64931	0.79453	0.82121	0.81957	0.84759	0.75013	0.79453

Cite this article as:
Nanwin, D.N., Agaji, I.,Ogala, E., & Gbaden, T.(2025). A framework for early detection of agile software development project failures. *FNAS Journal of Computing and Applications, 2*(2), 1-9.

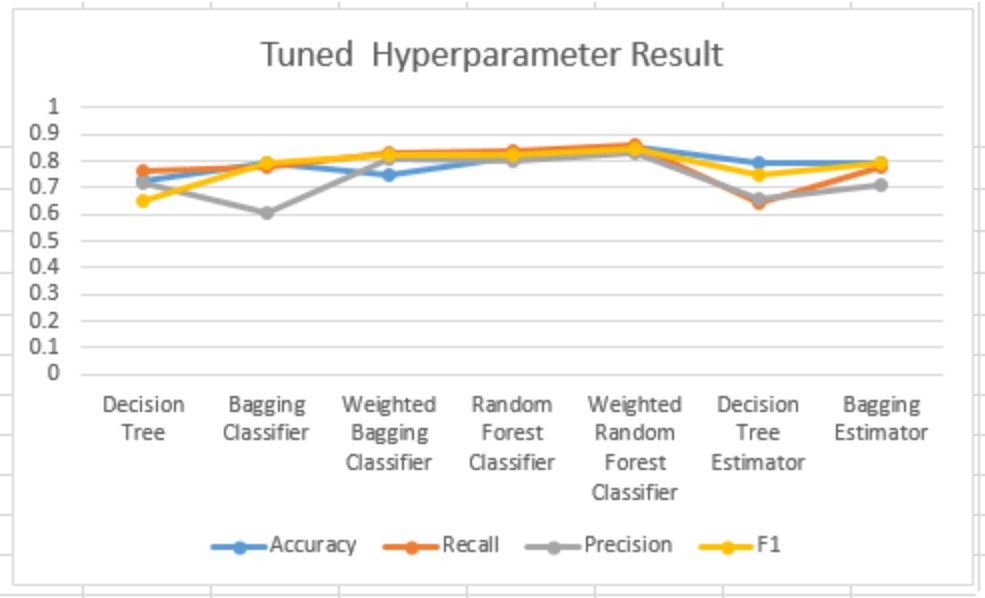

Figure 4: **Graphical Representation of the Hyper-parameter Tuned Result**

Table 2 presents the performance of various classification models after fine-tuning their hyper-parameters. The metrics evaluated include Accuracy, Recall, Precision, and F1 score. This performance is shown in the graphical representation in Figure 3. The Random Forest Classifier achieved the highest accuracy at 0.81364, indicating it correctly classified the most instances out of all the models. The Weighted Random Forest Classifier followed closely with an accuracy of 0.852, suggesting it performed even better than the standard Random Forest, likely due to the influence of weighting that improved its overall classification performance. The Decision Tree model had the lowest accuracy at 0.72190, which implies it was the least accurate among the models. The Weighted Random Forest Classifier achieved the highest recall at 0.85753, demonstrating its superior capability in identifying true positive instances. The Decision Tree Estimator exhibited the lowest recall at 0.64375, indicating it missed a significant number of relevant instances compared to the other models. The Weighted Bagging Classifier showed the highest precision at 0.81020, meaning it had the best performance in minimizing false positives. The Bagging Classifier had the lowest precision at 0.60854, suggesting it had more false positives compared to the other models and the Weighted Random Forest Classifier again demonstrated the best F1 score of 0.84759, indicating it achieved a strong balance between precision and recall. The Decision Tree had the lowest F1 score at 0.64931, reflecting its weaker performance in balancing precision and recall compared to other models.

Discussion

The Weighted Random Forest Classifier emerges as the most effective model for early detection of project failures, balancing the critical factors of precision, recall, and overall accuracy as seen in both Table 1 and Table 2. After fine-tuning the hyperparameters of the models, the provided metrics offer a detailed evaluation of their performance. A nuanced conclusion with respect to the framework for early detection of software development project failures has been presented. The Weighted Random Forest Classifier consistently performs the best across all metrics after fine-tuning. It excels in accuracy, recall, precision, and F1 score, making it the most robust model for detecting potential project failures early. This model effectively balances the trade-offs between detecting true positives (high recall) and ensuring prediction accuracy (high precision). Both the Random Forest Classifier and Weighted Bagging Classifier perform well but do not match the Weighted Random Forest Classifier in all metrics. The Random Forest Classifier shows strong performance, particularly in recall and F1 score, which is crucial for early failure detection. The Weighted Bagging Classifier has a high precision and recall but slightly lower overall accuracy compared to the Random Forest methods. The Bagging Classifier and Bagging Estimator are competitive but do not outperform the Random Forest methods. They offer a good balance of precision and recall but with lower accuracy compared to the best models. Both the Decision Tree and Decision Tree Estimator perform the least well across most metrics. Their lower F1 scores and less favorable recall and precision suggest that these models are less effective for the early detection of project failures. This could be due to their tendency to overfit or underfit compared to ensemble methods.

Cite this article as:
Nanwin, D.N., Agaji, I.,Ogala, E., & Gbaden, T.(2025). A framework for early detection of agile software development project failures. *FNAS Journal of Computing and Applications, 2*(2), 1-9.

The Weighted Random Forest Classifier overall outperformed other models across most metrics, including accuracy, recall, precision, and F1 score. This suggests that the weighting applied to the Random Forest model significantly enhanced its ability to classify instances accurately while balancing the trade-off between precision and recall. The Decision Tree model generally showed lower performance across all metrics, highlighting the limitations of a single decision tree in handling complex classification tasks compared to ensemble methods. The Bagging Classifier and Decision Tree Estimator showed mixed performance, with Bagging Classifier performing better in precision but slightly worse in recall and F1 score compared to Weighted Bagging Classifier. The Weighted Random Forest Classifier emerges as the most effective model for early detection of project failures, balancing the critical factors of precision, recall, and overall accuracy. It has the highest performance across all metrics after hyperparameter tuning. Incorporating these findings into the framework will help in selecting the most effective model for predicting software project failures, ultimately improving the chances of identifying and mitigating risks early.

Conclusion

The study presents an efficient framework for the early detection of failure in Agile software development projects. Through the integration of various machine learning algorithms and careful analysis of software project metrics, we have demonstrated the potential for proactive identification of project failures at different stages of development. Findings underscore the importance of incorporating failure detection analyses throughout the software development lifecycle, enabling timely interventions to mitigate risks and enhance project success rates. By leveraging ensemble learning techniques and comprehensive evaluation metrics, the work highlighted the strengths and limitations of different models in predicting project failure. This framework serves as a valuable tool for project managers and stakeholders in identifying potential pitfalls and optimizing project outcomes. Moving forward, further research is warranted to enhance the effectiveness and applicability of our framework. One avenue for future work involves refining the selection and integration of machine learning algorithms to improve predictive accuracy and robustness. Additionally, exploring additional metrics and features that may contribute to project failure prediction could enhance the framework's predictive capabilities. Furthermore, longitudinal studies tracking project outcomes over time could provide insights into the evolution of failure indicators and inform proactive risk management strategies. Moreover, conducting empirical studies in diverse software development contexts and industries would validate the generalizability and effectiveness of the framework. Overall, continued research in this area holds promise for advancing early detection techniques and improving the success rates of Agile software development projects.

References

Alonso, J. L., Belache, L., & Avresky, D. R. (2011). Predicting software anomalies using machine learning techniques, network computing and applications (NCA). *10th IEEE International Symposium on Software aging and rejuvenation.*

Charette, R. N. (2005). Why Software Fails “Software Failure”. IEEE Spectrum, 42(9), Pp. 42-49.

Chillar, D., & Sharma, K., (2019). Proposed T-Model to Cover 4S Quality Metrics Based on an Empirical Study of the Root Cause of Software Failures. *International Journal of Electrical & Computer Engineering, 9*(2).

Dauda, I. A., Nuhu, B. K., Abubakar, J., Abdullahi, I. M., & Maliki, D. (2021). Software Failures : A Review of Causes and Solutions. *ATBU Journal of Science Technology and Education (JOSTE), 9*(1), 415–423.

Eberendu, A. C. (2015). Evaluation of software project failure and abandonment in tertiary institutions in Nigeria. *Journal of Information and Knowledge Management, 5*(4).

Ebubeogu, F. A., & Lee, S. P. (2017). Integrated approach to software defect prediction. *IEEE Access, 5*, 21524–21547. doi:10.1109/access.2017.2759180

Egbokhare, F. (2014). Causes of Software/Information Technology Project Failures in Nigerian Software Development Organizations, 7, 107-110.

Janssen, N. E. (2019). A machine learning proposal for predicting the success rate of IT-projects based on project metrics before initiation *(Bachelor's thesis, University of Twente).*

Nevendra, M., & Singh, P. (2021). Software defect prediction using deep learning. *Springer International Journal on Deep Learning, 18*(10), 173 – 189.

Osegi, E. N., Anireh, V. I., & Onukwugha, C. G. (2018). PCWoT-MOBILE: A collaborative web based platform for real time control in the smart space. *iSTEAMS SMART-MIINDs Conference YABATECH, Lagos, Nigeria.*

Peng, H., Li, B., Liu, X., Chen, J., & Ma, Y. (2015). An empirical study on software defect prediction with a Simplified metric set. *Elsevier Journal of Information and Software Technology, 59*, 170-190.

Sommerville, I. (2011). Software engineering 9th Edition. *ISBN-10, 137035152*, 18.

Zhang, L., & Suganthan, P. N. (2014). Random forests with ensemble of feature spaces. *Pattern Recognition, 47*(10), 3429-3437.